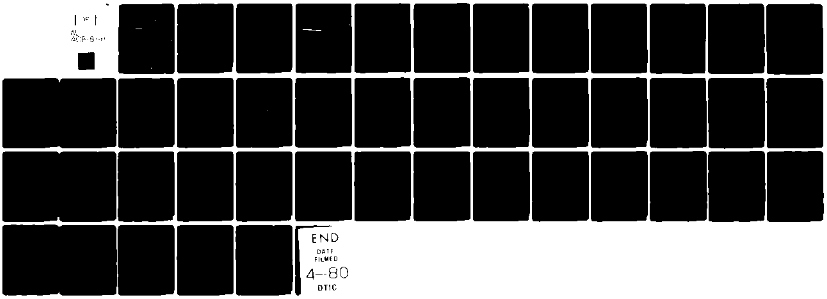
AD-A081 888 UNIVERSITY OF SOUTHERN CALIFORNIA MARINA DEL REY INFO--ETC F/G 5/7
COMPUTER AS AUTHOR -- RESULTS AND PROSPECTS.(U)
JAN 80 W C MANN, J A MOORE F49620-79-C-0181
UNCLASSIFIED ISI/RR-79-82 AFOSR-TR-80-0206 NL



END
DATE
FILMED
4-80
DTIC

ISI/RR-79-82
January 1980

William C. Mann
James A. Moore

Computer as Author—Results and Prospects

INFORMATION SCIENCES INSTITUTE
UNIVERSITY OF SOUTHERN CALIFORNIA
ISI
4676 Admiralty Way/ Marina del Rey/California 90291
(213) 822-1511

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER (19) (16) AFOSR-TR-80-0206	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) COMPUTER AS AUTHOR--RESULTS AND PROSPECTS		5. TYPE OF REPORT & PERIOD COVERED (7) Interim
		6. ~~PERFORMING ORG. REPORT~~ NUMBER
7. AUTHOR(s) (10) William C. Mann, James A. Moore		8. CONTRACT OR GRANT NUMBER(s) (12) new AFOSR F49620-79-C-0181
9. PERFORMING ORGANIZATION NAME AND ADDRESS USC/Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA 90291 (12) 43		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS (17) (16) 61102F 2304 A2
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Office of Scientific Research/NM, Bolling AFB, Washington, D.C. 20332		12. REPORT DATE (11) January 1980
		13. ~~NUMBER OF PAGES~~ 43
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) (14) ISI/RR-79-82		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

artificial intelligence, coherence, composition, computational linguistics, English, fragment-and-compose, knowledge delivery system, organization, synthesis, text, text generation.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

For a computer program to be able to compose text is interesting both intellectually and practically. Artificial Intelligence research has only recently begun to address the task of creating coherent texts containing more than one sentence.

One recent research has produced a new paradigm for organizing and expressing information in text. This raradigm, called Fragment-and-Compose, has been used in a pilot project to create texts from semantic nets. The method involves dividing the given body of information into many small propositional

20. Abstract cont.

units, and then combining these units into smooth coherent text. So far the largest example written by Fragment-and-Compose has been two paragraphs of instruction about what a computer operator should do in case of indications of a fire.

This report describes the text generation problem and anticipates a specific way to disseminate and use technical developments. It presents the research that led to creation of Fragment-and-Compose, including the largest example of computer-produced text. It also discusses the immediate problems and difficulties of elaborating Fragment-and-Compose into a general and powerful method.

11

ISI/RR-79-8
January 19

William C. Mann
James A. Moore

Computer as Author—Results and Prospects

INFORMATION SCIENCES INSTITUTE
UNIVERSITY OF SOUTHERN CALIFORNIA
4676 Admiralty Way/Marina del Rey/California 90291
(213) 822-1511

This research was supported in part by National Science Foundation grant No. MCS76-07332 and in part by the Air Force Office of Scientific Research contract No. F49620-79-C-0181. The participation of Neil Goldman and James Levin is gratefully acknowledged. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research of the U.S. Government.

Contents

Overview

Several distinct aspects of research on text synthesis are considered in this report. Each chapter was written separately for a distinct purpose. They are gathered here for convenience and because they jointly give a more comprehensive overview of this line of research.

Why and How?	Chapter 1 explores the motivations and potential benefits of text synthesis research. The present state of the art discriminates in favor of some styles of research; it discriminates against others. Attributes of a particularly timely style of research are identified, and the nature of research productivity in that style is described.
Where are we?	Chapter 2 presents our past work in text synthesis. This research produced both the KDS system and the Fragment-and-Compose paradigm. The chapter is an independent document made available here for the first time.
What next?	Chapter 3 considers the problems of going on to research that exploits the knowledge gained from KDS. It explores the issues of designing a successor system and identifies high priority problems for research attention.
Where to?	Chapter 4 is an attempt to anticipate a practical, useful form for results of the recent research and the immediately forthcoming research, of which the present project is a part. It is necessarily speculative, but it provides a target pattern of developments that can serve as both a bench mark and a draft for future developments.

1. Why Should a Computer be an Author?

Why, indeed? Computers are not known for their skills as authors, and little research has been done on causing them to write. Yet there is no fundamental reason for a machine to be incapable of creating high-quality text. The fact that machines are not authors reflects a profound ignorance of the technical details of the process of authoring (we avoid the more common term "writing" because it has already been consumed in other ways by computer technology), an essential process in producing most of mankind's intellectual products. Ignorance of its details (and even of its general nature) leads to great curiosity, which for some people is enough to spark interest in further research.

Computers have been used successfully to investigate many processes, some of which would not have been supposed to lend themselves to computer research, for example, speech perception and text comprehension. We should expect that attempting to make a computer function as an author, even in the most rudimentary way, would be an efficient discovery procedure, leading to new knowledge of the nature and details of the process of authoring. For the curious, an attempt to program an author is an expedition into uncharted regions, interesting and largely unexplored.

Beyond curiosity, there are practical motivations for pressing computers to create text. Natural languages, and only natural languages, have billions of readers. Natural language is often the only notation understood by all of the potential users of a collection of knowledge residing in a machine. And aside from the billions of potential readers, there are many people using computers today for whom computer output is obscure and frustrating. Computer output in their own language would help these people. For some of these computer users, modest advances in text composition techniques would be enough for practical purposes.

Practical motivations and scientific motivations usually lead to different kinds of research, but in this case they do not. The state of the art now easily permits the pursuit of both scientific knowledge and eventual usefulness through the same research.

2. Computer Generation of Multiparagraph English Text

2.1 Chapter Summary

This chapter reports recent research into methods for creating natural language text.[1] A new processing paradigm called Fragment-and-Compose has been created and an experimental system (KDS) implemented in it. The knowledge to be expressed in text is first divided into small propositional units, which are then composed into appropriate combinations and converted into text.

KDS (Knowledge Delivery System), which embodies this paradigm, has distinct parts devoted to creation of the propositional units, to organization of the text, to prevention of excess redundancy, to creation of combinations of units, to evaluation of these combinations as potential sentences, to selection of the best among competing combinations, and to creation of the final text. The Fragment-and-Compose paradigm and the computational methods of KDS are described.

2.2 Introduction

Computer users have difficulties in understanding what knowledge is stored in their computers; the systems have corresponding difficulties in delivering their knowledge. The knowledge in the machine may be represented in an incomprehensible notation, or we may want to share the knowledge with a large group of people who lack the training to understand the computer's formal notation. For example, there are large simulation programs that get into very complicated states we would like to be able to understand easily. There are data base systems with complex knowledge buried in them, but real problems in extracting it. There are status-keeping systems from which we would like to get snapshots. There are systems that try to prove things, from which we would like to have progress reports and justifications for various actions. Many other kinds of systems have knowledge-delivery difficulties.

The circumstances that make knowledge delivery in natural language particularly attractive are: a) complexity of the source knowledge, so that its notation is not easily

[1] This chapter has been submitted to a technical journal for publication.

learned, b) unpredictability of the demands for knowledge, so that the actual demands cannot be met with specific preprogrammed output, and c) the need to serve a large pool of untrained or lightly trained users of the knowledge.

For a number of the kinds of systems mentioned above, getting the information out is *one of the principal limitations on the system's uses.* If the information could be accessed more easily, then far more people could use the systems. So we're talking in part about facilitating existing systems, but much more about creating new opportunities for systems to serve people.

If computer systems could express themselves in fluent natural language, many of these difficulties would disappear. However, the necessary processes for such expression do not exist, and there are formidable obstacles even to designing such processes. The theory of writing is sketchy and vague, and there are few interesting computer systems to serve as precedents. Any research effort to create such systems--systems that know how to write--can be significant both in its practical implications and for the knowledge of writing that it produces.

Writing is an intellectually interesting task, though poorly understood. If we want to have a better theory, a better characterization of the nature of writing, then we can use computer program design and test as a discovery procedure for exploring the subject. In the present state of the art, the same research can create both theoretical knowledge and practical computational methods.

Of course, in a limited sense, programs already deliver knowledge in natural language by using "canned text." A person writes some text, possibly with the use of blanks, and the text is stored for use in association with a particular demand. The machine fills in the blanks as needed in a way anticipated as sufficient for the demand. This is a very useful technique, but it *does not* tell us much about the task of writing, and it does not generalize to situations in which the need for text has not been well anticipated in the programming stage. The basic organization and writing of canned text are done by a person, so that sort of "writing" *falls outside* the scope of this report.

The idea that natural language would be a useful output medium for a computer system is quite old, but the research to make such output possible is still in a very primitive state. Most of it has concentrated on sentence generation, in which problems such as word selection or translation to natural language from logical formulas in some propositional calculus have been chosen for study. These are important, but the needs for knowledge delivery simply cannot be satisfied by single sentences. It is necessary to have multiple sentences to communicate complex information, and multiple paragraphs would be even more valuable.

The remainder of this chapter thus focuses on computer generation of multisentence text. We present some examples of multisentence output from past systems and then discuss our own research.

2.3 Multisentential Text Generation in Past Research

Several systems have generated multisentence text;[2] sample texts from three of these are shown in Figure 1. One example is the SCHOLAR system, which describes aspects of South American geography. SCHOLAR's description of a South American country was several sentences long [Carbonell 73]. The SAM system, a story generator operating from a conceptual dependency script, also created multisentential output [Schank and project 75]. More recently, Meehan developed TALE-SPIN, a fable generator that worked in a different way, also based on a conceptual dependency representation [Meehan 77]. All of these systems generate multisentence text.

The systems share several features. First, the data structures that are the basis of the generation were designed for text processing; many of the special demands of text processing were anticipated and accommodated in the design of the knowledge structures themselves. Second, the sentence boundaries in these systems were direct correlates of internal features of the data structures themselves. Often the sentence order arose in the same way. Third, these systems had fixed generation goals, implicit in the code. Fourth, goal-pursuit occurred only in the content of the material being generated from; the systems themselves did not operate on a goal-pursuit algorithm. Finally, none of these systems chose the particular sentences to use in their output on the bases of quality assessment or comparisons among alternatives.

[2] All of the cited research has worked with English, but other languages would have served as well.

SCHOLAR:

```
*TELL ME ABOUT THE LOCATION OF URUGUAY.*

     THE LOCATION IN URUGUAY IS SOUTH AMERICA.
       THE LATITUDE RANGES FROM -30 TO -35 DEGREES.
       THE LONGITUDE RANGES FROM -53 TO -58 DEGREES.
       THE NORTHERN AND EASTERN BORDERING COUNTRY IS BRAZIL.
       THE WESTERN BORDERING COUNTRY IS ARGENTINA.
        THE BOUNDARY IS THE URUGUAY RIVER.
       THE SOUTHEASTERN COAST IS THE ATLANTIC OCEAN.
       THE SOUTHERN COAST IS THE RIO DE LA PLATA.
```

SAM:

John decided he was going to a restaurant. He went to one. He asked the hostess for a table. She told him he could go to it. He went to it. She went to it. He sat down in a chair. He got a menu from her. He read it. A waiter saw John was at the table. The waiter went to it. John ordered a lobster. A chef cooked it. The waiter got it from the chef. The waiter served it to John quickly. He ate it. He asked the waiter for a check. John got it from the waiter. John remembered the waiter had served the lobster to him quickly. John left the waiter a large tip. John paid the check. He left the restaurant.

TALESPIN:

ONCE UPON A TIME GEORGE ANT LIVED NEAR A PATCH OF GROUND. THERE WAS A NEST IN AN ASH TREE. WILMA BIRD LIVED IN THE NEST. THERE WAS SOME WATER IN A RIVER. WILMA KNEW THAT THE WATER WAS IN THE RIVER. GEORGE KNEW THAT THE WATER WAS IN THE RIVER. ONE DAY WILMA WAS VERY THIRSTY. WILMA WANTED TO GET NEAR SOME WATER. WILMA FLEW FROM HER NEST ACROSS A MEADOW THROUGH A VALLEY TO THE RIVER. WILMA DRANK THE WATER. WILMA WASN'T THIRSTY ANY MORE.

GEORGE WAS VERY THIRSTY. GEORGE WANTED TO GET NEAR SOME WATER. GEORGE WALKED FROM HIS PATCH OF GROUND ACROSS THE MEADOW THROUGH THE VALLEY TO A RIVER BANK. GEORGE FELL INTO THE WATER. GEORGE WANTED TO GET NEAR THE VALLEY. GEORGE COULDN'T GET NEAR THE VALLEY. GEORGE WANTED TO GET NEAR THE MEADOW. GEORGE COULDN'T GET NEAR THE MEADOW. WILMA WANTED TO GET NEAR GEORGE. WILMA GRABBED GEORGE WITH HER CLAW. WILMA TOOK GEORGE FROM THE RIVER THROUGH THE VALLEY TO THE MEADOW. GEORGE WAS DEVOTED TO WILMA. GEORGE OWED EVERYTHING TO WILMA. WILMA LET GO OF GEORGE. GEORGE FELL TO THE MEADOW. THE END.

Figure 1. Some published multisentence text samples

In all five of these points, the KDS research contrasts with these previous efforts. We have worked with data structures not designed for text generation, the sentence boundaries we develop are not direct correlates of internal features of the data structures, there are explicit goals for the generation process to satisfy, the system itself pursues goals, and the final text is chosen through quality comparisons among alternative ways of saying things.

2.4 The Task for the Knowledge Delivery System

In the light of these considerations, the problem can be restated more specifically as follows:

Given

1. An explicit goal of knowledge expression,

2. A computer-internal knowledge base adequate for some non-text purpose, and

3. Identification of the parts of the knowledge base that are relevant to the goal,

the task is to produce clean, multiparagraph text, in English, which satisfies the goal.

2.5 The Partitioning Paradigm

When we have stated this task to AI workers familiar with natural language processing, with no further specification, they have expected a particular kind of solution. They say, "Well, there are some sentence generators around, but the given information structures are too large to be expressed in single sentences. Therefore what we need is a **method for dividing up the input structure into sentence-size pieces**. Then we can give the pieces to a suitable sentence generator and get the desired text."

This is the expected solution, and people will simply presume that it is the line of development being taken.

That approach, which we call the Partitioning paradigm for text generation, was used in all the systems described above. In the Partitioning paradigm, the generation task is simplified by features of the knowledge base:

1. The knowledge base data structures have features that indicate appropriate sentence boundaries, and

2. The collection of information appropriate to be expressed in an individual sentence is adjacent. That is, a process can access all of the information appropriate to be expressed in a single sentence by following the data structure, without being required to traverse information to be expressed in other sentences.

These conditions prevail (by design) in all of the systems described above, but they are not generally typical of information storage in computers. KDS, on the other hand, takes an entirely different approach to the problem.

Several inherent difficulties become apparent when we attempt to use partitioning.

1. **Missing adjacencies**--Since (by our problem definition) the knowledge comes from a structure not prestructured for the generation task, what is and what is not adjacent in the knowledge base may be quite arbitrary. We may wish to include several widely scattered items in a sentence, so that it is not possible to carve out a piece with those items in it at all. The adjacencies that we need in order to partition the structure into sentence-size parts may simply be absent.

2. **Intractable residues**--Even though we may be able to find some way to **start** cutting out sentence-size objects from the data structure, there is no assurance at all that we will be able to run that method to completion and carve the **entire** structure into sentence-size pieces. Think of the comparable problem of carving statues from a block of marble. We may be able to get one statue or several, but if every part of the original block must end up looking like a statue, ordinary carving methods are insufficient. The residues left after carving out the first few statues may be intractable. A comparable sort of thing can happen in attempting to partition data structures.

3. **Lack of boundary correlates**--In some ways the worst difficulty is that an arbitrary given data structure does not contain structural correlates of good sentence boundaries. One cannot inspect the data structure and tell in any way where the sentence boundaries ought to be. Along with the other difficulties, this has led us to reject the expected solution, the Partitioning paradigm, and to create another.

2.6 The Fragment-and-Compose Paradigm

Our solution comes in two steps:

1. Find methods for fragmenting the given data structure into little pieces, small propositional parts.

2. Find methods for composing good sentences and good paragraphs out of those little parts.

We call this the Fragment-and-Compose paradigm. It is interesting to notice other systems employing a Fragment-and-Compose approach--e.g., building construction, papermaking, and digestion. In each, one begins by producing small, easily manipulated objects much smaller than the desired end-product structures, and then assembles these into the desired end products in a planned, multistage way. For the block of marble, the comparable processes are crushing and casting.

We may not be very encouraged in our text generation task by such precedents. However, there are precedents much closer to our actual task. The task of natural language translation resembles in many ways the task of translating from a computational knowledge source (although it has a comprehension subtask which we lack). Consider the (annotated) quotation below from *Toward a Science of Translating* [Nida 64].

Determination of Equivalence (Faithful Translation)

> The process by which one determines equivalence (faithfully translates) between source and receptor languages is obviously a highly complex one. However, it may be reduced to two quite simple procedures:
>
> (1) "decomposition" of the message into the simplest semantic structure, with the most explicit statement of relationships; and
>
> (2) "recomposition" of the message into the receptor language.

The quotation is from Nida's chapter on translation procedures. Notice particularly the two steps: decomposition and recomposition, and the emphasis on simple, explicit semantic structures in the results of the decomposition.

It turns out that this is the central procedural statement of Nida's book, and the remainder of the book can be seen as giving constraints and considerations on how this decomposition and recomposition ought to take place. We have very good reasons here for expecting that Fragment-and-Compose is an appropriate paradigm for natural language knowledge delivery.

To give a sense of what can be done using Fragment-and-Compose, here is a piece of a machine-generated text about what happens when fire breaks out in the computer room.

> Whenever there is a fire, the alarm system is started, which sounds a bell and starts a timer. Ninety seconds after the timer starts, unless the alarm system is cancelled, the system calls Wells Fargo. When Wells Fargo is called, they, in turn, call the Fire Department.

2.7 Description of KDS

Figure 2 is a block diagram of KDS, which simply says that KDS takes in an *Expressive Goal* (*telling* what the text should accomplish relative to its reader) and also a pre-identified body of Relevant Knowledge in the notation of its source. The output is clean multiparagraph text that can satisfy the goal.

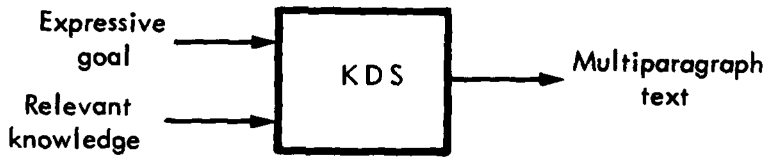


Figure 2. Input and output of KDS

We will be carrying a single example through this description of KDS. It is the most complex example handled by KDS, and it incorporates many ideas from previous studies on description of computer message systems.

A small contingency plans data base contains knowledge about what happens in various circumstances, and about people's actions, responsibilities, authorities, and resources. The particular knowledge to be delivered concerns a computer room in which there may be some indication of fire and in which there is a computer operator who should know what to do if that happens. This operator is the nominal reader of the text.

The general expressive goal is that the computer operator knows what to do in all of the predictable contingencies that can arise starting with an indication of fire. The contingencies are represented in the "Fire Alarm Scene," part of the knowledge base. A schematic sketch of the Fire Alarm Scene is given in Figure 3. (The figure is expository and contains far less information than the actual Scene. The Scene is a "semantic net," a collection of LISP expressions that reference the same objects.)

The knowledge identified as relevant includes not only the events of this scene but also enough information to support another computational task. In this case the knowledge is sufficient to support an alternate task, which we call the Motivation Exhibit task, that is to exhibit, for each action in the scene, the actor's reasons for performing the action. So, for example, the relevant knowledge includes the knowledge that fires destroy property, that destroying property is bad, that the amount of property destroyed increases with the duration of the fire, and that the Fire Department is able to employ methods for stopping fires. This is sufficient to be able to explain why the Fire Department attempts to stop fires. KDS does not perform the Motivation Exhibit task, but its knowledge is sufficient for it. We generate from a knowledge base sufficient for multiple tasks in order to explore the problems created when the knowledge representation is not designed for text processing.

The content of the scene is as follows:

> In the beginning state, INIT, the fire alarm sounds a bell. As we follow down the left side of the figure, we see that the fire alarm starts an interval timer, and at the end of the interval, the timer automatically phones Wells Fargo Company, the alarm system manager. Wells Fargo phones the fire department, and the fire department comes. The fire department fights the fire if there is one, and otherwise goes home.
>
> Meanwhile, the computer operator must pay attention to the alarm and decide what to do. He can block the alarm system's action, cancelling the alarm, or he can let the alarm system take its course. In the latter case, his next duty is to call the fire department himself, which has the same effect as Wells Fargo calling it. After that, his next duty is to flee. If he blocks the alarm then he is to go back to his previous task.

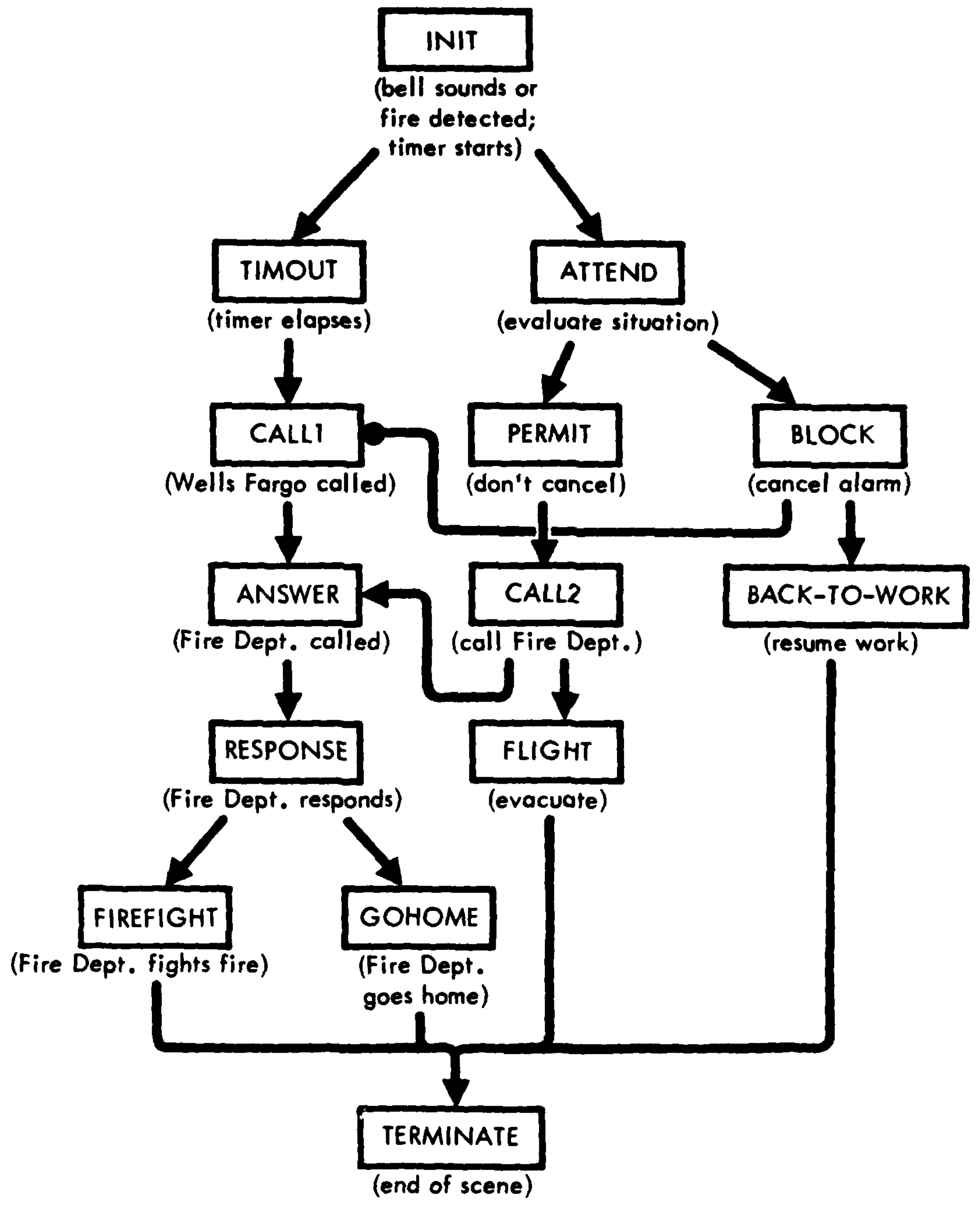


Figure 3. Events in the Fire-Alarm scene

2.8 Major Modules of KDS

KDS consists of five major modules, as indicated in Figure 4. A Fragmenter is responsible for extracting the relevant knowledge from the notation given to it and dividing that knowledge into small expressible units, which we call fragments or protosentences. A Problem Solver, a goal-pursuit engine in the AI tradition, is responsible for selecting the presentational style of the text and also for imposing the gross organization onto the text according to that style. A Knowledge Filter removes protosentences that need not be expressed because they would be redundant to the reader.

KDS MODULES	MODULE RESPONSIBILITIES
FRAGMENTER	• Extraction of knowledge from external notation • Division into expressible clauses
PROBLEM SOLVER	• Style selection • Gross organization of text
KNOWLEDGE FILTER	• Cognitive redundancy removal
HILL CLIMBER	• Composition of concepts • Sentence quality seeking
SURFACE SENTENCE MAKER	• Final text creation

Figure 4. KDS module responsibilities

The largest and most interesting module is the Hill Climber, which has three responsibilities: to compose complex protosentences from simple ones, to judge relative quality among the units resulting from composition, and to repeatedly improve the set of protosentences on the basis of those judgments so that it is of the highest overall quality. Finally, a very simple Surface Sentence Maker creates the sentences of the final text out of protosentences.

The data flow of these modules can be thought of as a simple pipeline, each module processing the relevant knowledge in turn. We will describe each of these modules individually.

2.9 Fragmenter Module

The Fragmenter takes in the relevant knowledge as it exists externally and produces a set of independent protosentences, called the Sayset. (See Fig. 5.) These primitive fragments, the protosentences, have no intended order. (In our experiments, they are presented in a list that is immediately randomized.) Each primitive protosentence can, if necessary, be expressed by an English sentence.

So the problem for the remainder of the system is to express well what can surely be expressed badly. It is important to note that this is an improvement problem rather than a problem of making expression in English feasible.

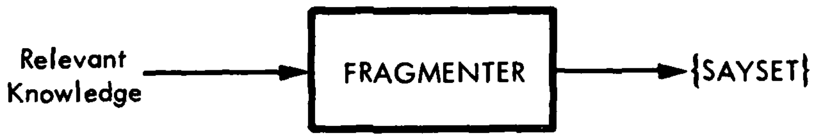


Figure 5. Fragmenter module input and output

The protosentences the Fragmenter produces are propositional and typically carry much less information than a sentence of smooth English text.

So, in our example, the fragmenter produces the list structures shown in part below for two of its fragments.

```
((CONSTIT (WHEN (CALLS NIL WELLS-FARGO)
                (CALLS WELLS-FARGO FIRE-DEPT)))...)

([CONSTIT (WHENEVER (STARTS NIL ALARM-SYSTEM)
                    (PROB (SOUNDS ALARM-SYSTEM
                                  BELL]...)
```

2.10 Problem Solver Module

The second major module is the Problem Solver (Fig. 6). The primary responsibilities of the Problem Solver are to select a text presentation style and to organize the text content according to the selected style. For this purpose, it has a built-in taxonomy of styles among which it selects. Although the taxonomy and selection processes are very rudimentary in this particular system, they are significant as representatives of the kinds of structures needed for style selection and style imposition.

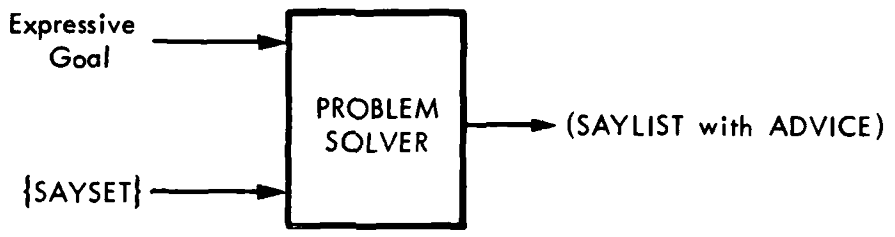


Figure 6. Problem Solver input and output

We believe that text style should be selected on the basis of the expected effects. In simple cases this is so obvious as to go unrecognized; in more complex cases, which correspond to complex texts, there are many stylistic choices. In order to select a style, one needs:

1. A description of the effect the text should have on the reader,
2. Knowledge of how to apply stylistic choices, and
3. A description of the effects to be expected from each stylistic choice.

Note that these are required whether stylistic choices are distributed or wholistic, i.e., whether they are made in terms of attributes of the final text or in terms of particular methods for creating or organizing the text.

The first item, a description of desired effects, is (more or less by definition) a goal. The second item is the set of applicable methods, and the third is the knowledge of their effects. The Problem Solver is a goal-pursuit process that performs means-ends analysis in a manner long familiar in AI. The information organization is significant partly because of the demand it puts on the knowledge of style: Knowledge of style must be organized according to expected effect.

The Problem Solver takes in the Sayset produced by the Fragmenter and the Expressive Goal given to the system and produces a Saylist, which is an ordered list of the protosentences, some of which have been marked with Advice. The Problem Solver pursues given goals. It has several submodules that specialize in particular kinds of goals, including modules Tell and Instructional-narrate, which are active on this example. The Problem Solver can operate on the current Saylist with three kinds of actions in any of its modules:

1. It can Factor the Saylist, extracting all protosentences with a particular character or attribute and placing them before all those that lack that attribute, retaining order within each sublist. The result is a simple list consisting of the upper sublist, a new paragraph-break protosentence, and the lower sublist.

2. It can impose an order on some or all of the elements of the Saylist.

3. It can mark protosentences with Advice. Sometimes the Problem Solver knows some attribute of the final text that ought to be achieved, perhaps because of a demand of the chosen style, but it has no way to effect this directly. In this case it marks all the affected protosentences with Advice, which will be acted on after the Problem Solver has finished.

Figure 7 below describes the rules used in the Problem Solver that carry out these three kinds of actions. In this example, the Tell module acts before Instructional-narrate.

The first Tell rule corresponds to the heuristic that the existence of something ought to be mentioned before its involvement with other things is described. The third rule corresponds to the heuristic that the writer (KDS) ought to reveal its own goals of writing before pursuing those goals.

Instructional-narrate uses a presentational technique that makes the reader a participant in the text. So, for example, the final text says, "When you hear the alarm bell ...," rather than "When the operator hears the alarm bell...," Instructional-narrate knows that the role of "you" should be emphasized in the final text, but it has no direct way to achieve this. To every protosentence that refers to "you," it attaches advice

Factoring Rules

TELL

1. Place all (EXISTS ...) propositions in an upper section.
2. Place all goal propositions in an upper section.
3. Place all writer's goal propositions in an upper section.

INSTRUCTIONAL-NARRATE

1. Place all propositions with non-reader actor in an upper section.
2. Place all time dependent propositions in a lower section.

Ordering Rules

INSTRUCTIONAL-NARRATE

1. Order time-dependent propositions according to the (NEXT ...) propositions.

Advice-giving Rules

INSTRUCTIONAL-NARRATE

1. YOU is a good thing to make explicit in the text.

Figure 7. Rules used in the Problem Solver

saying that explicit reference to the reader, which is done by mentioning "you" in the final text, has positive value. This advice is taken inside the Hill-climber.

In our example the Problem Solver creates the following fragment:

```
(PARAGRAPH-BREAK (REASON: (BOUNDARY NON-H-ACTOR)))


((CONSTIT (WHEN (IF (POSSIBLE)
                    (CALL YOU FIRE-DEPT))
                (EVOKE YOU EVAC-SCENE)))...
  (ADVISORS FRAG INST-NARRATE)
  (ADVICE ...(GOOD YOU)))
```

2.11 Knowledge Filter Module

The Knowledge Filter is a necessary part of KDS because as soon as we attempt to create text from a knowledge base suitable to support some other computational purpose, we find a great deal of information there that ought not to be expressed, because the reader already knows it.

This is a general phenomenon that will be encountered whenever we generate from an ordinary computational knowledge base. As an illustration, consider Badler's work on getting a program to describe a movie in English.

Figure 8 is reproduced from [Badler 75]. It shows fifteen successive scenes from a short computer-generated movie. The graphics system that generates the movie provides a stock of propositional knowledge about it. The objects in the scene are known to the machine unambiguously and in sufficient detail to generate the movie. The research task is to create a computer program that will describe in English the physical activities in this and similar movies. The detail is voluminous, and so Badler is faced with a *serious information suppression* problem. After several stages of applying various filtering heuristics, such as "Don't describe directly anything that doesn't move," he can represent the movie by the five statements below.

C.1 There is a car.

C.2 The car starts moving toward the observer and eastward, then onto the road.

C.3 The car, while going forward, starts turning, moves toward the observer and eastward, then northward-and-eastward, then from the driveway and out-of the driveway, then off-of the driveway.

C.4 The car, while going forward, moves northward-and-eastward, then northward, then around the house and away-from the driveway, then away-from the house and stops turning.

C.5 The car, while going forward, moves northward, then away.

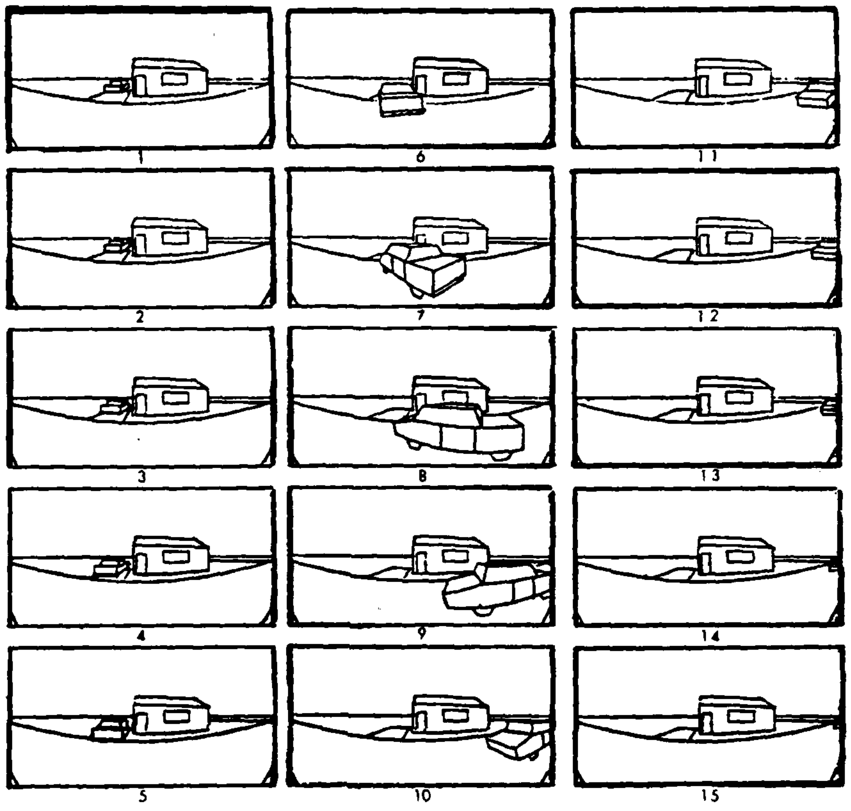


Figure 8. Badler's "Moving Car Scenario"

These are still too cumbersome, so additional stages of reduction are applied, yielding the single statement:

> The car approaches, then moves onto the road, then leaves the driveway, then turns around the house, then drives away from the house, then stops turning, then drives away.

Even the longer text above contains only a fraction of the available information about the car and the other objects. Information on their types, their subparts, visibility, mobility, location, orientation and size are available from Badler's source. He also develops a sequence of events to describe the movie, based on certain indicators of continuity and discontinuity. The volume of information available, the predictability of its parts, and the insignificance of some of its details are such that all of it could not have been expressed in a smooth text.

One of the principal activities of Badler's system is selection of information to be removed from the set of knowledge to be expressed. Some things need not be expressed because they follow from the reader's general knowledge about motion of objects; others are removed because they represent noisiness, rather than significant events, generated by the processes that discern motion.

The point for us is simply that **the demands of smooth text production are incompatible with expression of all of the available information.** Text production requires condensation and selectivity, the process we call knowledge filtering, on any reasonably complete body of knowledge. Knowledge filtering is a significant intellectual task. It requires coordinated use of a diversity of knowledge about the reader, the knowledge to be delivered, and the world in which all reside. We now recognize the necessity of sophisticated knowledge filtering as part of the process of producing quality text.

KDS's Knowledge Filter inputs the Saylist, including Advice, from the Problem Solver, and outputs the Saylist with additional Advice, called "Don't Express" advice, on some of the protosentences. (See Fig. 9.) So some of the items have been marked for omission from the final text. (They are marked rather than deleted so that they are available for use if needed as transitional material or to otherwise make the resulting text coherent.) It decides which protosentences to mark by consulting its internal model of the reader to see whether the propositional content is known or obvious. Although KDS's reader model does not contain any inference capabilities about what is obvious, a more robust model certainly would. We recognize that the work of the Knowledge Filter is a serious intellectual task, and we expect that such a filter will be an identifiable part of future text creation programs.

In our example the Knowledge Filter produces the DON'T-EXPRESS advice in the following element of the Saylist:

```
((CONSTIT (WHENEVER (SOUNDS NIL ALARM-BELL)
                    (HEARS YOU ALARM-BELL)
                    (PROB)))...
 (ADVISORS INST-NARRATE NONEXP)
 (ADVICE (GOOD YOU)
         DON'T-EXPRESS))
```

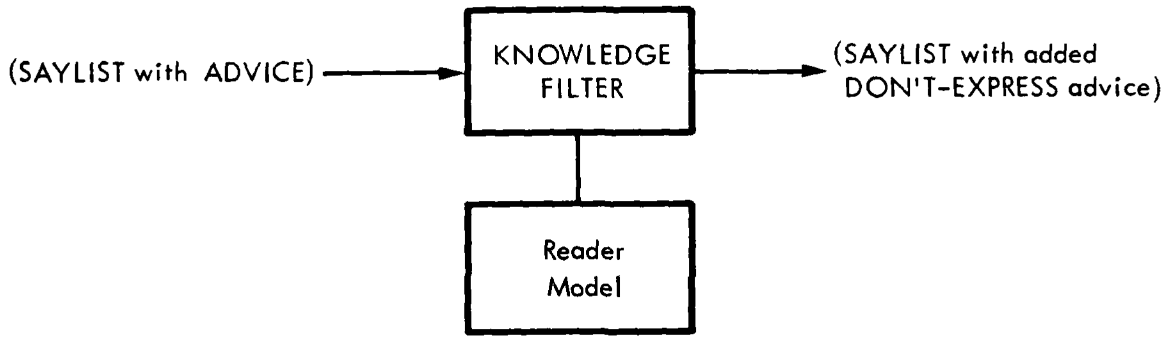


Figure 9. Knowledge Filter module input and output

2.12 Hill Climber Module

The Hill Climber module (Fig. 10) consists of three parts:

1. A somewhat unconventional hill-climbing algorithm that repeatedly selects which one of an available set of changes to make on the Saylist.

2. A *set of Aggregation rules* (with an interpreter) telling how the protosentences may legally be combined. These correspond roughly to the clause-combining rules of English, and the collection represents something similar to the writer's competence at clause coordination. Each Aggregation rule consumes one or more protosentences and produces one protosentence. Advice propagates onto the protosentences produced.

3. A set of Preference rules (with an interpreter) able to assign a numerical quality score to any protosentence. The score computation is sensitive to Advice.

The algorithm is equivalent to the following. Scores are assigned to all of the primitive protosentences, then the Aggregation rules are applied to the Saylist in all possible ways to generate potential next steps up the hill. The resultant

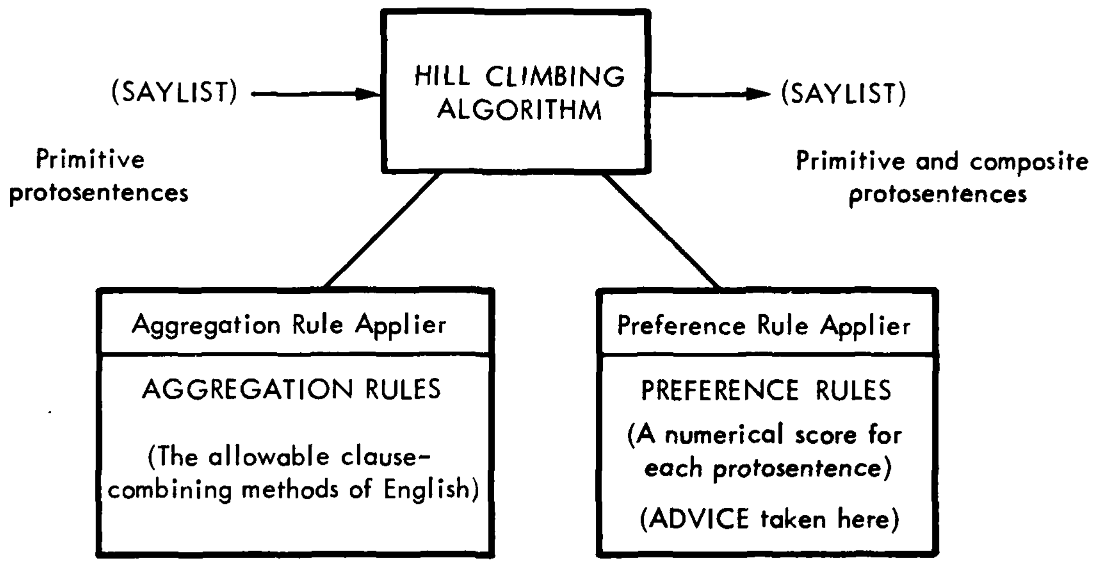


Figure 10. Hill Climber module

protosentences are also evaluated, and the Hill Climber algorithm then compares the scores of units consumed and produced and calculates a net gain or loss for each potential application of an Aggregation rule. The best one is executed, which means that the consumed units are removed from the Saylist, and the new unit is added (in one of the positions vacated, which one being specified in the Aggregation rule).

This process is applied repeatedly until improvement ceases. The output of the Hill Climber is a Saylist for which there are no remaining beneficial potential applications of Aggregation rules.

This Saylist improvement activity is the technical heart of the text production process, it develops the final sentence boundaries and establishes the smoothness of the text.

Figure 11 shows a few of the Aggregation rules. (Each of them has been rewritten into an informal notation suggesting its content.) Aggregation rules are intended to be meaning-preserving in the reader's comprehension, but are not intended to preserve explicitness.

```
1. Common cause.
        Whenever C then X.
                                 Whenever C then X and Y.
        Whenever C then Y.

2. Conjoin mid-state
        Whenever X then Y.
                                 Whenever X then Y and then Z.
        Whenever Y then Z.

3. Delete mid-state
        Whenever X then Y.
                                 Whenever X then Z.
        Whenever Y then Z.

4. Delete existential
        There is a Y.
        <mention of Y>           <mention of Y>
        (Y is known unique)

5. If-then-else
        If P then Q.
                                 If P then Q otherwise R.
        If not P then R.

6. Test and branch
        When P then determine whether X.
        If X then Q.             When P then determine X and
        If not X then R.         decide Q or R.
```

Figure 11. Sample Aggregation rules

These are only a few of the Aggregation rules that have been used in KDS; others have been developed in the course of working on this and other examples. Coverage of English is still very sparse. In other examples, an aggregation rule has been used to produce a multiple-sentence structure with intersentential dependencies.

Figure 12 shows the Preference rules.

One of the surprising discoveries of this work, seen in all of the cases investigated, is that the task of text generation is dominated by the need for brevity: How to avoid saying things is at least as important as how to say things. Rule 1 introduces a tendency toward brevity, because most of the Aggregation rules consume two or three protosentences but produce only one, yielding a large gain in score. Sentences produced from aggregated protosentences are generally briefer than the corresponding sentences for the protosentences consumed.

1. Every protosentence gets an initial value of -1000.

2. Every primitive protosentence embedded in a composite protosentence decreases its value by 10.

3. If there is advice that a term is good, each occurrence of that term increases value by 100.

4. Each time-sequentially linked protosentence after the first increases value by 100.

5. Certain constructions get bonuses of 200: the if-then-else construct and the When-X-determine-Y.

6. Any protosentence produced by multiple applications of the same aggregation rule gets a large negative value.

Figure 12. Preference rules

Rule 3 introduces the sensitivity to advice. We expect that this sort of advice taking does not need to be elaborate--that being able to advise that a term is good or a term is bad is adequate.

Rule 6 is somewhat of a puzzle. Empirically, a sentence produced by reapplication of an Aggregation rule was always definitely unacceptable, primarily because it was awkward or confusing. We do not understand technically why this should be the case, and some say it should not be. We do know that this rule contributes significantly to overall quality.

The selection algorithm of the Hill Climber is somewhat unconventional in that it does not select the Aggregation rule application with the largest increase in collective score, which would be the usual practice. The hill of collective scores has many local maxima, which can be traced to the fact that one application of an aggregation rule will preclude several others. Because protosentences are consumed, the various applications are in competition, and so a rule that produces a large gain may preclude even more gain.

The *Hill Climber* selects the rule application to use based on an equation that includes competitive terms. It computes the amount of gain surely precluded by each application and makes its selection on the basis of maximum net gain, with the precluded gain subtracted.

2.13 Sentence Generator Module

The Sentence Generator (Fig. 13) takes the final ordered set of protosentences produced by the Hill Climber and produces the final text, one sentence at a time. Each sentence is produced independently, using a simple context-free grammar and semantic testing rules. Because sentence generation has not been the focus of our work, this module does not represent much innovation, but merely establishes that the text formation work has been completed and does not depend on further complex processing.

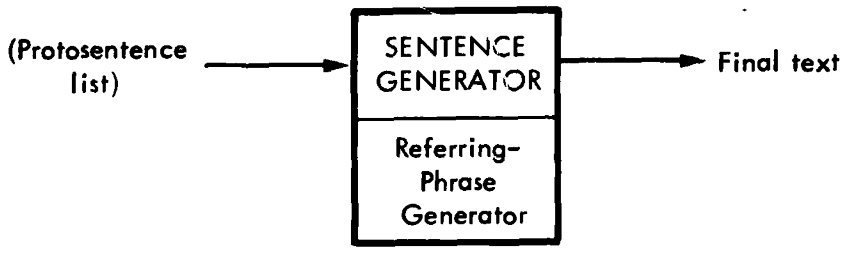


Figure 13. Sentence Generator module input and output

The single significant innovation in the Sentence Generator is the Referring Phrase Generator, the only part in which prior sentences affect the current sentence. The *Referring Phrase* Generator keeps track of what objects have been referred to, and how. It presumes that objects previously referred to are in the reader's attention and that after they have been identified by the first reference, subsequent references need only distinguish the object from others in attention. This process is equivalent to the one described by [Levin and Goldman 78] developed for this research. It knows how to introduce terms, refer to objects by incomplete descriptions, and introduce pronouns. However, none of our examples has exercised all of the features of Levin and Goldman's algorithm.

2.14 Output Text

Applying all of this machinery in our example, we get the result shown in Figure 14. Note the paragraph break, a product of a factoring rule (the first rule in Instructional-narrate) in the Problem Solver module.

> Whenever there is a fire, the alarm system is started, which sounds a bell and starts a timer. Ninety seconds after the timer starts, unless the alarm system is cancelled, the system calls Wells Fargo. When Wells Fargo is called, they, in turn, call the Fire Department.
>
> When you hear the alarm bell or smell smoke, stop whatever you are doing, determine whether or not there is a fire, and decide whether to permit the alarm system or to cancel it. When you determine whether there is a fire, if there is, permit the alarm system, otherwise cancel it. When you permit the alarm system, call the Fire Department if possible, then evacuate. When you cancel the alarm system, if it is more than 90 seconds since the timer started, the system will have called Wells Fargo already, otherwise continue what you were doing.

Figure 14. Final fire-alarm text from KDS

2.15 Conclusions and Prospect

The development of KDS highlights several aspects of the task of writing that strongly influence text quality. The overwhelming importance of brevity, seen in both the Knowledge Filter and the Preference rules, is striking. Writing is seen here as a constructive activity rather than simply as interpretive. That is, it is not so much a mapping between knowledge representations as it is the creation of new symbolic objects, not equivalent to older ones, but suitable for achieving particular effects. The image of writing as a kind of goal pursuit activity is helpful in factoring the task into parts. The task (and the program) is occupied with finding a good way to say things, not with establishing feasibility.

The KDS development has also established features of the knowledge-delivery program design problem. The defects of the Partitioning paradigm are newly appreciated; the Fragment-and-Compose paradigm is much more manageable. It is easy to understand, and the creation of Aggregation rules is not difficult. The separation of Aggregation and Preference actions seems essential to the task, or at least to making the task manageable. As a kind of competence/performance separation it is also of theoretical interest. Knowledge filtering, as one kind of responsiveness of the writer to the reader, is essential to producing good text.

The importance of fragmenting is clear, and the kinds of demands placed on the Fragmenter have been clarified, but effective methods of fragmenting arbitrary knowledge sources are still not well understood.

In the future art, we expect to see the Fragment-and-Compose paradigm reapplied extensively. We expect to see goal-pursuing processes applied to text organization and style selection. We expect distinct processes for aggregating fragments and selecting combinations on a preference basis. We also expect a well developed model of the reader, including inference capabilities and methods for keeping the model up to date as the text progresses. Finally, we expect a great deal of elaboration of the kinds of aggregation performed and of the kinds of considerations to which preference selection responds as well.

3. Problems in Knowledge Delivery

How can the Fragment-and-Compose paradigm be developed into a generally effective approach to knowledge delivery? What are the problems to overcome in doing so? What kinds of goals should guide immediate efforts to develop techniques within this paradigm?

One kind of goal arises from the nature of the task, and a different kind from the details of its current state of development. The first kind is discussed below; the second in section 3.1.

We have chosen to focus on the task of delivering knowledge in the form of multisentence English text. What is the general condition of the state of the art for such *design? It has several prominent features:*

1. Every kind of relevant information is scarce--abstract principles, prior system designs, working precedents, useful algorithms are all hard to find.

2. Existing precedents (of any of these kinds) tend to be specialized to the particular environments in which they arose, and so they tend to depend on arbitrary combinations of conditions that occur only infrequently.

3. There is no theoretical framework useful to designers.

4. There are no widely accepted criteria for judging the quality of knowledge delivery. However, subjective judgements of relative quality for the sorts of text that machines can produce are usually easy and uncontroversial. They are an adequate guide to quality effects.

Advancing this state of the art calls for an approach focusing on:

- invention of new methods rather than evaluation of existing methods,

- feasibility rather than economy,

- informal research style rather than formal style,

- delivering knowledge in new ways rather than refining existing ways,

- generalization of methods wherever possible.

The current state of the art thus puts high value on invention and innovation. Research in knowledge delivery must therefore present a flexible framework that can accommodate a great deal of unpredictable change. It must address problems definite enough to stimulate and receive innovations.

Notice that these priorities arise from the general condition of the art, not from the details of its technical content. There are also important priorities that come from the technical content; they are discussed in section 3.1.

If possible, the research framework should suggest ways to generalize the innovations it creates. In practical terms, this means that the research should not be carried out in a theoretical vacuum.

Even though there is no suitable theoretical framework that the designer of a knowledge delivery system may use, there are relevant theoretical perspectives for the research. The function of these perspectives is to suggest generalizations of innovations. They give a way of thinking about the relationships between a method that works in a particular case and the range of cases in which that method could be made to work. The perspectives particularly important to this research are:

- Philosophy
    * Speech Act Theory
    * Theory of Action
    * Logic
- Linguistics
    * Syntax
    * Semantics
    * Pragmatics and Discourse
- Computer Science
    * Artificial Intelligence
    * Natural Language Processing

The limitations of the Fragment-and-Compose paradigm are not well understood. We cannot predict the difficulties that might arise with a new domain of knowledge to deliver, a new representational notation from which to generate, or new kinds of uses for the generated text. These kinds of extension are all largely unexplored.

It is therefore important to exercise the paradigm on a diversity of knowledge delivery tasks, tasks chosen to search out the strengths and weaknesses of the paradigm as a whole.

3.1 Issues Raised by the Fragment-and-Compose paradigm

The Fragment-and-Compose paradigm can be divided into seven parts:

1. Knowledge Location
2. Fragmenting
3. Planning
4. Knowledge Filtering
5. Aggregation
6. Preference Determination
7. Sentence Generation

Since there are design issues for every element of the paradigm, we need to identify the issues both technically significant and influential on progress and immediately addressable by research. The sections below identify and discuss such problems.

3.1.1 Problems of Knowledge Location

The KDS system was given a goal to pursue and a pre-identified body of "relevant" knowledge useful in pursuing the goal. The issue of how to find the body of relevant knowledge was not addressed. We intend that the Fragment-and-Compose paradigm be usable in systems not designed with text generation in mind. (We call this outer system the "source system" because it is the source of the knowledge delivered.) The source system cannot be expected to deliver pre-identified bodies of relevant knowledge on demand.

We therefore need a new kind of subprocess, which we call a Locator, to actively discover the knowledge to be delivered in response to a goal. The Locator is not analogous to any process in KDS.

The information representation in the source system is expected to be idiosyncratic, incomplete, and not organized for the knowledge delivery task. Since relevance is inevitably somewhat uncertain, the Locator is expected to be inclusive, to identify the possibly-relevant and exclude the great bulk of surely-irrelevant knowledge.

The Locator must determine relevance with respect to a goal expression that is one of its inputs. How can an item of knowledge be relevant to a goal?

> For the given goal, there are several Methods that potentially could satisfy the goal, and for each Method there are several Preconditions and Input Conditions that govern its use, and Methods may use other Methods as subordinates. An item of knowledge is sufficiently relevant for the purposes of the Locator if it satisfies any of the following conditions:
>
> 1. It tells (entirely or in part) whether a Precondition is satisfied;
> 2. It tells (entirely or in part) whether an Input Condition is satisfied;
> 3. It is relevant to a Method that is a potential *subordinate* of a Method known to be usable.

In other words we may attempt to anticipate every kind of information use that could take place in pursuing the given goal and decide that all such information is relevant.

This approach is unsatisfactory for several reasons:

- It is not selective enough, since it delivers all of the information that appears relevant to all of the potential methods for responding to the goal.
- It is disorganized, since it does not necessarily associate an item of information delivered with the various methods for using the information.
- It is inefficient, since in general it produces far more information than can be used.

The concept of "relevance" carried forward from KDS is not precise enough for our task. The Locator should deliver information that will be useful to the methods **actually employed** by the system. However, there is a potential circularity: selecting methods involves knowing which methods are feasible, which in turn involves knowing what information is available, and locating information for the method involves prior selection of a method.

One way out of the circularity is to associate with each method an independent test of its feasibility. A feasibility test might be inexpensive relative to the cost of locating all of the information that would potentially be used by a method. This observation leads to a research interest in identifying independent feasibility tests for a system's principal methods of expressing knowledge.

This means that the role of the Locator will not be that of an independent preprocessor, as would be suggested by KDS, but rather that the Locator should operate in close interaction with the Planning activities, as Planning searches through the space of feasible plans to construct an appropriate plan.

3.1.2 Problems of Fragmenting

The Fragmenter works closely with the Locator, translating information from an idiosyncratic external notation to a notation suitable for all of the further processing. The Fragmenter's output notation (representation) must be particularly strong. It should be possible to represent:

- all of the "facts" in the present source system;
- facts in other knowledge domains that might be represented in other source systems;
- combinations of facts arising in building complex sentences;
- various speech acts that can be performed relative to such facts.

Much of the generality and future utility of a knowledge delivery system thus depends on the robustness of the fragment representation scheme. In a sense, we want it to have the full expressive power of natural language, since complex protosentences will be encoded in it. On the other hand, it must be subject to a wide variety of formal operations.

3.1.3 Problems of Planning

The "planning" operations include organizing the text and selecting a suitable style. The chief research problem in planning is to discover a useful and manageable decomposition of these operations. In general there is a vast amount of knowledge of text styles in the world, but it is widely acknowledged that good writing requires invention as well. This large body of information and techniques is well beyond the reach of today's research. What is needed is the command of a much more modest range of skills. The KDS system provided for a range of skills and document types only in principle, since the modules not exercised by the fire-response task were mere placeholders. It remains for a multisentence text generation system to embody any significant enumeration of potential styles.

3.1.4 Problems of Knowledge Filtering

The function of the Knowledge Filter is to mark knowledge that, if expressed, is likely to be useless to the hearer.[3] The principal way that knowledge may be useless is by already being mutually known by the hearer and the system. Such knowledge is "obvious" to the hearer, and it is marked by the Knowledge filter so that such material can be given special treatment (usually deletion). The difficulties all arise in knowing what is "obvious."

Several inadequate views should be set aside immediately:

1. What is obvious is (to a good approximation) a fixed list that we can create for use with any particular hearer or audience. But what is obvious is also combinatorially unmanageable. If we must search a given list to find that it is obvious to all readers of ISI technical reports that all Russian admirals have livers, we are lost.

2. What is obvious is that which is known (and perhaps in attention at the moment) and that which necessarily follows from it. But logical necessity does not lead to obviousness. Consider the four-color theorem.

   Conversely, some things are obvious that do not necessarily follow from what is known. It is "obvious" that anyone who speaks English fluently knows what a pencil is.

3. For a particular hearer, every proposition is either obvious or not, so that what is obvious to the hearer can be represented by a predicate on hearers and propositions, say OBVIOUS(H,P). Unfortunately, what is not obvious in one circumstance and time is obvious in another, and what is not obvious may suddenly become so. Obviousness depends not only on available information and circumstances, but also on the time available for reflection.

[3] We follow the linguists convention of referring to speaker and hearer, regardless of whether the media of communication include an auditory one.

The knowledge filter must include some model of the hearer. It must include an inferential capability in order to identify the "obvious" but unenumerable facts, but it must not simply be an inference engine that will identify that which is necessarily so. Designing this limited inference capability so that it represents real hearers well is currently the principal technical problem of knowledge filtering.

In the long run, there will be other kinds of uselessness for which we will want identification and knowledge filtering action. Uselessness is a lack of function. The system will come to contain a positive theory of the functions of knowledge, since it will be using knowledge to achieve goals. The other kinds of knowledge filtering will tend to be complementary to this theory. We expect that some future generation of systems would use the same theory for both text planning and identifying other kinds of candidates for deletion from being expressed. Addressing such problems is premature.

3.1.5 Problems of Aggregation

Aggregation is in essence the competence to put knowledge into combinations that can be expressed as single sentences in English. It includes all of the available variations, without regard for their frequency in use. There are some complex aggregation structures. One moderately complex one is built around the word "respectively." Some, such as the "whereas" structures of legal language, can be used iteratively at indefinite length. Brevity, parallelism, and the effects of repetition all involve forms of aggregation.

KDS had an interesting but very small set of aggregation rules. They were autonomous and meaning-preserving.

One part of the immediate challenge in aggregation is to discover and make explicit a useful working set of aggregation forms. They are easy to discover in existing text, but it is not clear how many are needed for fluent expression.

A second part of the challenge is more difficult. One might imagine that the effect of expressing a combination of items of knowledge would be the same as the combined effects of expressing the items individually. This kind of additive property prevails in logic and mathematics, but there are many information-combining forms of expression in English that are not additive.

Consider the combination

(a) John is rich.

(b) John is stupid.

vs.

(c) John is rich but stupid.

The latter expresses a contrast by using "but," suggesting that there is some surprise or incongruity found in the combination of wealth and stupidity. The suggestion is missing when the two propositions about John are expressed separately, e.g., by

(d) John is rich and stupid.

It might be difficult to convey the suggestion of incongruity more succinctly than by this use of "but."

The point is simply that the aggregation forms have consequences beyond merely expressing their component items of knowledge. It is essential to understand these consequences, to associate them with the aggregation forms that produce them, and to design systems so that they produce these effects appropriately.

What are these nonadditive effects? Many of them can be seen as affecting the coherence of the text. There are many varieties of coherence, many notions of continuity in text that affect the hearer. Brian Phillips, in [Phillips 79], mentions these:

1. Anaphora (2 kinds)
2. Spatial coherence
3. Temporal coherence
4. Causal coherence
5. Thematicity

There are many other potentially useful analyses of coherence [Halliday 76].

The first four areas are particularly fertile in producing effects of combination. Anaphora, temporal sequence, causal consequences, and spatial connection all have special words for expressing them, words frequently used to combine assertions. In addition, mere juxtaposition can be used to express these relations as well as others.

This leads to a serious technical challenge, the design of an aggregation-rule representation and interpreter to accommodate the nonadditive effects of aggregations and yet operate with the autonomy and meaning-preserving character of the KDS rules.

3.1.6 Problems of Preference Determination

The function of preference determination is to select, among all of the currently feasible aggregation operations, one which will be applied. The design of a preference determiner is complicated by several inherent difficulties:

1. There is no clear standard of preference, and people's preferences vary and differ a great deal.
2. Even for a particular individual, context, situation and time, there are some equally preferable alternatives that do not lead naturally to a suitable choice.
3. The factors leading people to prefer one expression over another are poorly known and hard to discern, even for the individual who has a definite preference.
4. Selectivity in text production may reflect accidental systematic factors as well as genuine preference. Hobbs has supported this view.

Beyond those factors, development of preference processes depends on maturity of the processes providing the alternatives. It is therefore not timely to attempt a sophisticated development of preference notions and processes. The general scheme represented in KDS can be extended in detail to be adequate for present research purposes.

3.1.7 Problems of Sentence Generation

The sentence generator of KDS was a postprocessor that operated on a list of aggregated propositions. It did not participate in the system's selectivity, and none of the consequences of sentence generation were directly available for preference determination. These are design flaws of KDS. They can be corrected in part by changing the control structure so that sentence generation is a resource for other parts of the system rather than a postprocessor.

The sentence generator in KDS was rudimentary; it did not require innovations outside of those required for repeated reference [Levin and Goldman 78], and even the developments for repeated reference were not fully utilized. We expect that sentence generation for multiparagraph text will not strain the state of the art in the forthcoming systems. A more flexible structure than that of KDS's sentence generator is needed, as much for research convenience as for its theoretical significance.

3.2 Summary of Problems

We recognize a need for new knowledge delivery methods to enable computer systems to communicate to their users, especially in natural language. In pursuit of this goal a new paradigm of text-generation, called Fragment-and-Compose, has been created. Initial experience with this paradigm suggests that it has the appropriate flexibility and generality for development into a general-purpose knowledge delivery technique. This effort might include developing text-generation methods and programs that are reusable, sharable and adequate for new tasks with the provision of a localized system-specific knowledge location part.

The state of the art is currently weak. Useful knowledge and experience are scarce, and many of the specific problems of knowledge delivery have not yet been addressed. This puts a premium on invention of new techniques and new concepts. Theory is currently more useful in evaluating innovations than in designing a knowledge delivery system.

The current state of development of Fragment-and-Compose raises several technical challenges that can be addressed directly. Prominent among these are:

- the design of general-purpose knowledge location processes;

- a control structure for efficient locator control;

- a robust representation for primitive and aggregated fragments;

- a limited-inference capability with output that approximates "obviousness to people";

- substantial coverage of the aggregation forms of English;

- systematic use of nonadditive aggregation forms, with explicit provision for coherent results.

4. Natural Language Knowledge Delivery in the Future

Although knowledge delivery processes are usually designed on an ad hoc, system-by-system basis, it is worthwhile to have a systematic method in view for the future when the problem and possible solutions will be better understood. This chapter presents such a method, with the expectation that making this vision explicit will help us to develop and modify it until it becomes a realistic pattern of design and implementation.

In this paradigm, natural language knowledge delivery is a capacity that can be routinely designed for a system. There are many routine programming tasks today for which there are no serious issues of feasibility, no major uncertainties about the effectiveness of available techniques. Such tasks as building an assembler for a new computer instruction set, providing geometric data analysis to surveyors, writing a sorting program, or providing a text editor for a new timesharing system may be interesting or even challenging, but feasibility is well established. We can be sure from the outset that an acceptable solution can be found. In this sense, they are routine.

BEGIN FUTURE

The designer of a knowledge delivery capability has available existing program modules corresponding to the Problem-solver, Knowledge Filter, Aggregator, Preference Determiner and Sentence generator of KDS. These supply a basic generating capability for English, with a basic vocabulary, text organization and paragraph smoothing capabilities. The notation used in the Knowledge Filter is widely circulated and well understood. The User Model comes with a stock of common knowledge and inference rules for deciding what is "obvious." The part of the User-model holding specific declarations about the users of the new system is incomplete, so the implementer must add to the representation of what the user knows.

The implementer must create a new Fragmenter for his system, which can transfer propositional knowledge from the given representation in the new system to the standard "protosentence" representation used in the delivery modules. He must also create a Locator module that can access the system's knowledge in its own representation and identify the parts relevant for expression in response to a given formal Goal. Finally, as part of the Locator, he must provide one or more drivers for the knowledge delivery subsystem. Each driver is a source of formal Goals that tell the knowledge delivery subsystem what to accomplish.

All of the parts the implementer supplies are specific to the new task and system. All of the given modules are independent of task and system, but specific only to the particular natural language chosen. Modules are available for English and certain related European languages, and several sets may be used for multilingual output.

In this design paradigm, development of the modules that know natural language is cumulative, rather than being fresh for each new system. This knowledge is independent of both the system's tasks and its knowledge representation, since knowledge of these is localized in the system-specific Fragmenter, Locator, and Drivers.

Part of the User-model also arises in a cumulative fashion. The User-model contains two kinds of knowledge, a) common knowledge and inference rules that arise in every system (such as knowledge of the months of the year, or of how to deduce which of two events came first) and b) task-specific and system-specific knowledge, and knowledge about the particular users being addressed at the moment. These are not cleanly separable, but the implementer can expect that prior implementers who have utilized this User-model have supplied nearly all of the common knowledge needed.

To summarize, the implementer's problem of supplying natural language knowledge delivery is, module by module:

Sentence Maker	ACQUIRE FROM PREVIOUS USE
Aggregator	ACQUIRE FROM PREVIOUS USE
Preference Determiner	ACQUIRE FROM PREVIOUS USE
Knowledge-Filter	ACQUIRE FROM PREVIOUS USE
User-model	ACQUIRE COMMON KNOWLEDGE FROM PREVIOUS USE
---	---
	DECLARE USER'S TASK KNOWLEDGE
Fragmenter	PROGRAM IT
Locator	PROGRAM IT

END FUTURE

Although we recognize that this implementer's paradigm has not yet been achieved, it functions as a guideline for planning and as a partial factoring of the knowledge delivery program into independent communicating modules. Making it a reality appears to be technically feasible and manageable within the conventional system-development methods currently used. We expect that in the future, reuse of a particular set of knowledge generator modules can be as easy as reuse of compiler, assembler, and loader modules is today.

References

[Badler 75] Badler, N.I., "The conceptual description of physical activities," in *Proceedings of the 13th Annual Meeting of the Association for Computational Linguistics*, 1975. AJCL Microfiche 35.

[Carbonell 73] Carbonell, J. R., and A. M. Collins, "Natural semantics in artificial intelligence," in *Proceedings of the Third International Joint Conference on Artificial Intelligence*, pp. 344-351, 1973.

[Halliday 76] Halliday, M. A. K., and R. Hasan, *English Language Series*. Volume 9: *Cohesion in English*, Longman, London, 1976.

[Levin and Goldman 78] Levin, J.A., and N. M. Goldman, *Process Models of Reference in Context*, USC/Information Sciences Institute, Research report 78-72, 1978.

[Meehan 77] James R. Meehan, "TALE-SPIN, an interactive program that writes stories," in *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, 1977.

[Nida 64] Eugene Nida, *Toward a Science of Translating*, E. J. Brill, Leiden, 1964.

[Phillips 79] Phillips, B., "A model for knowledge and its application to discourse analysis," *American Journal of Computational Linguistics* Microfiche 82, 1979.

[Schank and project 75] Schank, Roger C., and the Yale A.I. Project, *SAM--a story understander*, Yale University, Dept. of Computer Science, Research report 43, 1975.